File System Implementation

Disk Structure

partitions: disk can be subdivided into partitions

raw usage: disks/partitions can be used raw (unformatted) or formatted with file system

volume: entry containing FS
— tracks that file system's info is in device directory or volume table of contents

FS diversity: there are general purpose and special purpose FS

File Systems — layers

layer 5: applications

layer 4: logical file system

layer 3: file-organization module

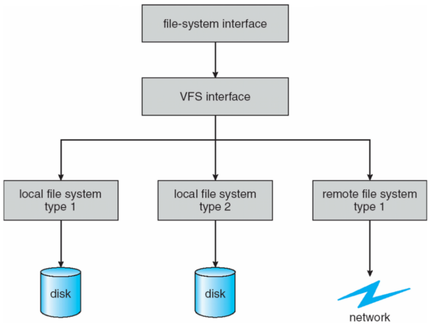
layer 2: basic file system

layer 1: I/O control

layer 0: devices

File Systems — virtual

principle: provide object-oriented way of implementing file systems
— same API used for different file system types



Files — implementation

meta data must be tracked:
— which logical block belongs to which file?
— block order?
— which blocks are free for next allocation?

block identification: blocks on disk must be identified by FS (given logical region of file)
⟶ meta data needed in *file allocation table*, *directory* and *inode*

block management: creating/updating files might imply allocating new/modifying old disk blocks

Allocation — policies

preallocation:
— *problem*: need to know maximum file size at creation time
— often difficult to reliably estimate maximum file size
— users tend to overestimate file size to avoid running out of space

dynamic allocation: allocate in pieces as needed

Allocation — fragment size

extremes:
— fragment size = length of file
— fragment size = smallest disk block size (= sector size)

trade-offs:
— *contiguity*: speedup for sequential accesses
— *small fragments*: larger tables needed to manage free storage and file access
— *large fragments*: improve data transfer
— *fixed-size fragments*: simplifies space reallocation
— *variable-size fragments*: minimizes internal fragmentation, can lead to external fragmentation